Encoding Application Profiles in a Computational Model of the Crosswalk

Carol Jean Godby
OCLC, USA
godby@oclc.org

Devon Smith
OCLC, USA
smithde@oclc.org

Eric Childress
OCLC, USA
childress@oclc.org

Abstract

OCLC's Crosswalk Web Service (Godby, Smith and Childress, 2008) formalizes the notion of *crosswalk*, as defined in Gill,et al. (n.d.), by hiding technical details and permitting the semantic equivalences to emerge as the centerpiece. One outcome is that metadata experts, who are typically not programmers, can enter the translation logic into a spreadsheet that can be automatically converted into executable code. In this paper, we describe the implementation of the Dublin Core Terms application profile in the management of crosswalks involving MARC. A crosswalk that encodes an application profile extends the typical format with two columns: one that annotates the namespace to which an element belongs, and one that annotates a 'broader-narrower' relation between a pair of elements, such as Dublin Core *coverage* and Dublin Core Terms *spatial*. This information is sufficient to produce scripts written in OCLC's Semantic Equivalence Expression Language (or Seel), which are called from the Crosswalk Web Service to generate production-grade translations. With its focus on elements that can be mixed, matched, added, and redefined, the application profile (Heery and Patel, 2000) is a natural fit with the translation model of the Crosswalk Web Service, which attempts to achieve interoperability by mapping one pair of elements at a time.

Keywords: application profiles; Dublin Core; Dublin Core Terms; semantic interoperability; MARC; metadata crosswalks

1. Application Profiles and Metadata Mapping

A preservation society in Ohio has just digitized some old photographs of Chillicothe, the state capital from 1803 until 1810 and the home of Majestic Theater, which has operated continuously for over a century and a half and has hosted many famous vaudeville performers, including Laurel and Hardy and Milton Berle. To make these images accessible to students and local history buffs, volunteers create a Dublin Core description that includes a title, description, and subject for each image, which renders them visible to automated harvesting utilities. But since this is a curated set of images about a particular place, the description could be enhanced with a record that describes the entire collection, using vocabulary from the Dublin Core Collection (DCMI, 2007) application profile, which includes a statement about access rights, pointers to associated collections, and a description of how the collection is accrued.

An *application profile* is a "declaration of the metadata terms an organization, information resource, application, or user commuity uses in its metadata," according to Greenberg and Severiens (2007), and is motivated by the need to enhance the discovery of a resource by diverse groups of people. In our hypothetical but realistic example, the owners of the images want to make their resources accessible to students or the curious public in a way that also preserves a piece of the historical record for future scholars. At the 2007 International Conference on Dublin Core and Metadata Applcations, project leaders from four continents reported on the design and use of application profiles to serve similar needs. For example, the SCROL (Singapore Cultural Resources Online) project designed a profile for managing access to images from multiple databases controlled by museums and archives (Wu, et al, 2007). And the DRIADE project (Digital Repository of Information and Data for Evolution) developed a profile for the

management of heterogeneous data relevant to the study of evolutionary biology (Carrier, et al, 2007).

Though these projects have achieved varying degrees of technical maturity, most acknowledge the seminal work of Heery and Patel (2000), who characterize the application profile as a formalism that resolves the conflict between two groups of stakeholders. On the one hand, standards developers want to encourage consistency and continuity; on the other, application developers require flexibility and responsiveness. To meet the needs of both groups, Heery and Patel describe guidelines for the creation of application profiles, which may:

- *Draw on one or more existing namespaces.* Technically, a namespace is an element defined in an XML schema, though it is often understood to refer to a named domain containing a list of terms that could be, but is not yet, expressed in a formal syntax. In our scenario, elements such as *title* or *description* belong to the Dublin Core namespace, while elements such as *accrual method* belong to the Dublin Core Collections namespace. Additional namespaces can be added if they are required for a more detailed description. For example, if the digitized photos are used in a high-school course on the history of Ohio, the description might be enhanced with an element such as *audience* from the Gateway to Educational Materials (GEM, 2008) namespace, whose value would specify that this resource is appropriate for high-school juniors and seniors.
- *Refine standard definitions—but only by making them narrower, not broader*. For example, the GEM *audience* element is intended to annotate the grade level of a resource that can be used in a classroom. But since *audience* is a specialized description, it is formally linked to *description,* an element defined in the Dublin Core namespace that can replace it when a less detailed record is required. Because of this restriction on how definitions can be refined, the application profile permits complementary operations on the elements that comprise it. An element is *refined* or *replaced* when the element with the narrower meaning substitutes for the corresponding broader one. And an element is *dumbed down* when the element with the broader meaning is used instead.
- *Introduce no new data elements.* Data elements may not be added to existing namespaces, but may only be introduced into a description by including more namespaces, as we've indicated. To extend our example, suppose the historical society needed to keep track of where the records describing the digitized images reside in a local database. If so, a metadata standards expert could define a namespace such as *ChillicotheHistoricalSociety*, which might contain a *database-id* element, and add it to the application profile.

The technical infrastructure of the application profile addresses the needs of standards makers by creating incentives to use existing descriptive frameworks instead of creating new ones, preserving some degree of interoperability among records that describe similar resources. For systems designers, the application profile permits complex descriptions to be built up or collapsed using easily formalized operations.

In this paper, we show how the machinery of the application profile defined by Heery and Patel aids in the efficient management of metadata formats that are translated to and from MARC in OCLC's Crosswalk Web service (Godby, Smith and Childress, 2008), a utility that powers the metadata translation functions in OCLC Connexion® Client and a growing number of other products and services. The focus of our effort is the relationship between MARC and Dublin Core Terms (hereafter, DC-Terms) (DCMI, 2008), a namespace and de-facto application profile that extends Unqualified Dublin Core (hereafter DC-Simple) by adding elements such as *AudienceLevel* or *Mediator* and by refining DC-Simple elements such as <dc:relation> with *isReferencedBy* or *isReplacedBy*.

To implement the relationship between MARC and DC-Terms, we need to solve three problems. First, since the only publicly accessible crosswalk (LOC, 2008) was last updated in 2001 and has been defined only for one direction, from MARC to DC-Terms, we need to

expedite the process of acquiring translation logic from metadata standards experts and converting it to executable code. Second, we need to manage versions. Software that exploits the inheritance structure in an application profile can process records conforming to DC-Terms and DC-Simple schemas, or to these schemas with local extensions. Given that MARC can be extended in similar fashion, and that input or output records may have multiple structural realizations—as XML, ISO-2709, or RDF, among others—the number of record variants to be translated can quickly explode. Finally, we need to manage change because standards are always evolving, as are the use cases that invoke them.

The solutions to all three problems emerge from the fact that the application profile, as well as the translation model underlying the Crosswalk Web Service, focus on the goal of achieving element-level interoperability, as described in the recent surveys by Chan and Zeng (2006) and Zeng and Chan (2006). The resulting is strikingly simple. The metadata subject matter expert edits a spreadsheet, from which the corresponding executable code is automatically generated. As a consequence, about two dozen types of records can be processed in a software environment that is rarely touched by human hands, eliminating a software maintenance problem in the MARC-to-Dublin Core crosswalk with a model that can eventually be applied to other relationships.

2. The DC-Terms application profile in the Crosswalk Web Service

Figure 1 illustrates the process flow for the translation of a record by the Crosswalk Web Service. As shown at the top of the figure, the input is a small (and invalid) MARC record consisting of a single field and subfield, *522 a Northwest*, encoded either in the MARC XML (LOC, 2007b) or ISO-2709 (ISO, 2008) syntax. The output is an XML-encoded DC-Terms record containing the element *DCTerms:spatial*, shown at the bottom of the figure. In Step 1, a utility program that we call a *reader* converts the native MARC input to a standardized, easy-to-process XML container syntax that we call Morfrom. Step 2 translates this record to a Morfrom representation of DC-Terms. In Step 3, a utility called a *writer* converts this result to an output syntax (here, another XML encoding) that has been formally defined by the Dublin Core standards community for DC-Terms.

These are the major operations of the core business logic in the Crosswalk Web Service. For a more technical discussion of the processing and data models, as well as the arguments that motivate this design, the reader is referred to our most recent article (Godby, Smith and Childress, 2008).

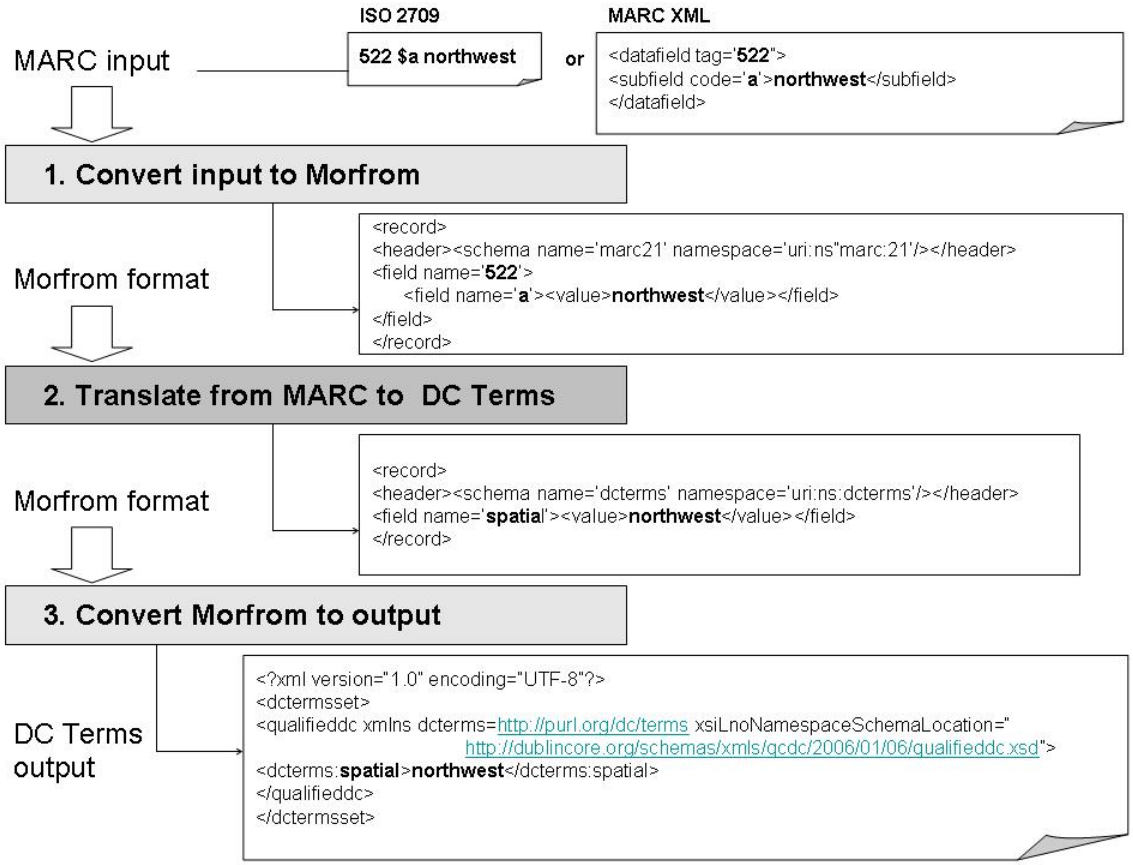


FIG. 1. Data flow in the Crosswalk Web Service.

Since our present focus is on application profiles, the most important step in the model is Step 2, where the translation from MARC to Dublin Core Terms takes place. Three critical issues must be addressed. How does the metadata subject matter expert communicate the translation logic to the implementation? How is the translation implemented? And how does the application profile interact with the translation?

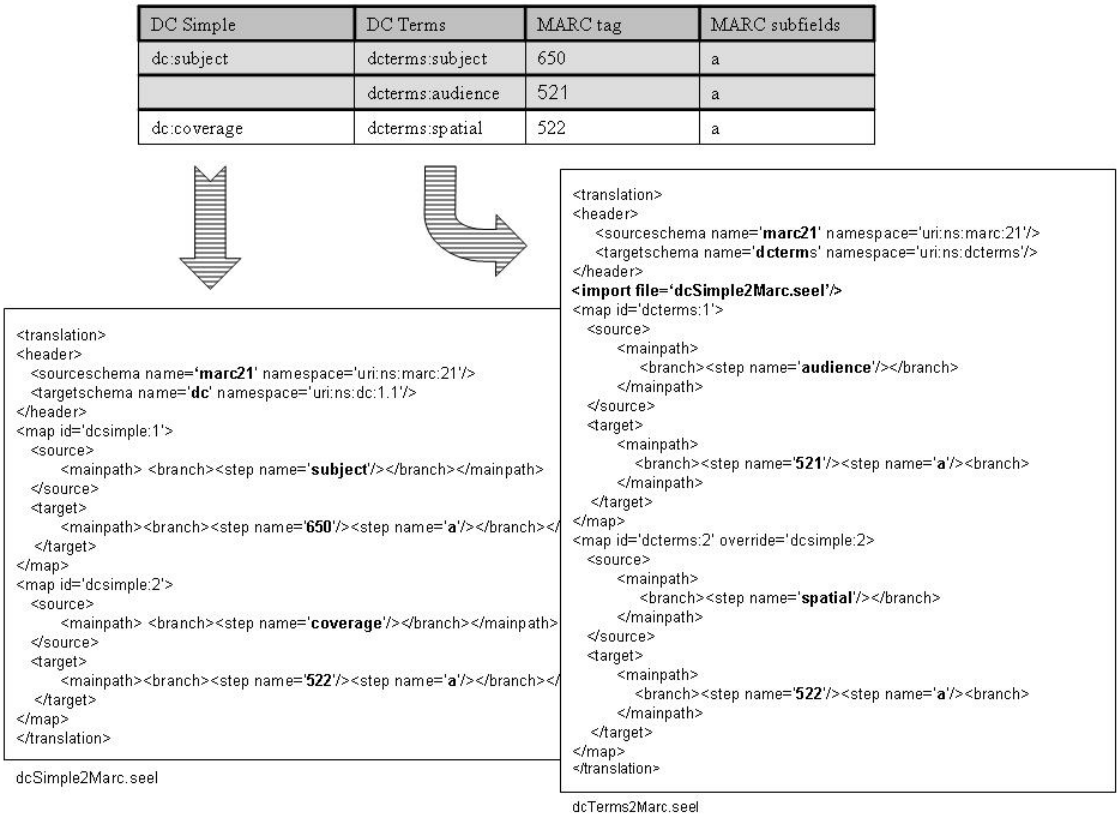
To answer the first question, the metadata standards expert fills out a spreadsheet like the one shown at the Figure 3, which shows the most important elements for implementing a crosswalk involving an application profile when the corresponding elements are related by a straightforward lexical substitution. Such is the case for our sample record shown in Figure 1, where *522 a* in the input is replaced by *spatial* in the output, a relationship that is expressed in the last row of the table in Figure 3. (More subtle relationships can also be expressed, as we will discuss shortly.) In the same row, the standards expert has recorded two additional facts about the Dublin Core-to-MARC relationship: that *dc:coverage* also maps to MARC 522 a, and that *dcterms:spatial* is 'dumbed down' to *dc:coverage* when a DC-Terms record requires a Dublin Core Simple manifestation. Thus, if the user with the record shown in Figure 1 had specified an output of DC-Simple instead of DC-Terms, the result would have been the record shown in Figure 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<simpledc xmlns dc='http://purl.org/dc/elements/1.1/>
<dc:coverage>northwest</dc:coverage>
</simpledc>
```

FIG. 2. A DC-Simple record.

The other rows in Figure 3 contain less complex information and are greyed out because they mention elements that are not represented in our sample record. In the first two rows, the names for the DC-Simple and DC-Terms are the same, so any effect of the dumb-down operation would be invisible. In the third row, *dcterms:audien*ce is not mapped to any corresponding DC-Simple element because it represents an element definition in the DC-Terms namespace that extends the descriptive scope of DC-Simple.

After the standards expert has filled out or edited the crosswalk spreadsheet, a software developer runs a Perl script against it to produce executable code. A sample is shown in the two boxes at the bottom of Figure 3. This is Seel (or Semantic Equivalence Expression Language) code, which we have designed, along with a program that interprets it, to model the information found in a typical crosswalk and make it actionable. A Seel script, expressed in XML, corresponds to a translation. The most important elements are <map>, which is a self-contained representation of a single row in a crosswalk; <source>, which identifies the input element, in this example, *522 a*; and <target>, which identifies an output element such as *coverage* or *spatial*. The <mainpath> element defines a path in the Morfrom record where the data of interest is located. In our sample record, the data *northwest* is found in a path ending with the elements *522 a* and will be written to a path containing elements named *coverage* or *spatial*. Finally, in addition to a set of maps, a Seel script must have a <header>, which lists locally defined URIs where technical specifications for the source and target schemas can be resolved.



DC Simple	DC Terms	MARC tag	MARC subfields
dc:subject	dcterms:subject	650	a
	dcterms:audience	521	a
dc:coverage	dcterms:spatial	522	a

```
<translation>
<header>
  <sourceschema name='marc21' namespace='uri:ns:marc:21'/>
  <targetschema name='dc' namespace='uri:ns:dc:1.1'/>
</header>
<map id='dcsimple:1'>
  <source>
      <mainpath> <branch><step name='subject'/></branch></mainpath>
  </source>
  <target>
      <mainpath><branch><step name='650'/><step name='a'/></branch></
  </target>
</map>
<map id='dcsimple:2'>
  <source>
      <mainpath> <branch><step name='coverage'/></branch></mainpath>
  </source>
  <target>
      <mainpath><branch><step name='522'/><step name='a'/></branch></
  </target>
</map>
</translation>
```

dcSimple2Marc.seel

```
<translation>
<header>
   <sourceschema name='marc21' namespace='uri:ns:marc:21'/>
   <targetschema name='dcterms' namespace='uri:ns:dcterms'/>
</header>
<import file='dcSimple2Marc.seel'/>
<map id='dcterms:1'>
  <source>
      <mainpath>
         <branch><step name='audience'/></branch>
      </mainpath>
  </source>
  <target>
      <mainpath>
         <branch><step name='521'/><step name='a'/><branch>
      </mainpath>
  </target>
</map>
<map id='dcterms:2' override='dcsimple:2>
  <source>
      <mainpath>
         <branch><step name='spatial'/></branch>
      </mainpath>
  </source>
  <target>
      <mainpath>
         <branch><step name='522'/><step name='a'/><branch>
      </mainpath>
  </target>
</map>
</translation>
```

dcTerms2Marc.seel

FIG. 3. A spreadsheet format and corresponding Seel maps.

Both Seel scripts shown in Figure 3 are automatically generated when the Perl script is applied to the spreadsheet. Both Seel files are complete scripts in the sense that they can be executed 'as is,' but a useful translation would usually contain many more maps than than those shown here. The script on the left is called whenever a client requests DC-Simple output from the Crosswalk Web Service. The script on the right is called when users ask for DC-Terms records.

Before we proceed with the discussion, it is instructive to highlight the most important features of these translations. The first script contains the DC element *subject*, which does not appear in the second. The second script contains a map to the DC-Terms element *audience*, which does not appear in the first. Both scripts contain maps to the DC elements from MARC 522 that represent geopatial information—*coverage* and *spatial*—at different levels of granularity. These maps and their relationships to each other in the two translations are sufficient to illustrate the essential operations that must be modeled in the execution of an application profile in a translation. How is an element dumbed down? How is an element added to a namespace? How is the meaning of an element in one namespace overridden by an element in another? And how is an instance record conforming to the application profile, which must, by definition, contain a mixture of elements from different namespaces, produced in a translation?

Conceptually, the dumb-down operation is the easiest. To map *coverage* from MARC 522 a instead of *spatial*, the client needs only to run the dcSimple2MARC script (on the left in Figure 3) instead of dcTerms2MARC. To dumb down an original Dublin Core Terms record instead of a MARC record, the user submits DC-Terms input to the Crosswalk Web Service and specifies DC-Simple as the output. Internally, the Web service would translate DC-Terms to MARC and then MARC to DC-Simple using the corresponding translations from the other direction that have been generated in the same manner as those we discuss here.

The other operations are implemented through elements defined in the Seel language. To describe how maps to elements in multiple namespaces are added to the translation and how the translated record ends up with a mixture of elements from different sources, we need to point out a detail that appears in the DC-Terms script but is not present in the DC-Simple script. The <import> element, shown in bold type in Figure 3, forces the inclusion of the maps defined in the DC-Simple script to create a comprehensive translation that consists of maps from both scripts. One effect of the *impor*t operation is to extend the maps involving the base standard—here, DC Simple—with a set of elements from a different namespace. In our example, the map to *dcterms:audience* appears in the DC-Terms output as the only new extension. But the spreadsheet and corresponding Seel script that represent the full application profile contains many more elements coded in this pattern.

Another effect of the <import> element on the DC-Terms script is to propagate into the translated record the binding of the *audience* element to the DC-Terms namespace and that of the *subject* element to the DC-Simple namespace. Recall that the <header> element in the DC-Terms script specifies *dcterms* as the namespace of the target, ensuring that the *dcterms* namespace is attached to the topmost element and is inherited by the target elements of each map in the translation—here, *spatial* and *audience*. An analogous operation happens when the DC-Simple script is executed. But when dcSimple2Marc.seel is imported into the DC-Terms script, the DC-Simple namespace is explicitly attached to every element that appears in DC-Simple but not in DC-Terms by a local *namespace* attribute, which overrides the default *dcterm*s namespace that would have otherwise been assigned. Below is a DC-Terms record that contains the elements *audience* and *subject*. The Morfrom representation of a record produced by the application of the Seel scripts in Figure 3 is shown first (with the explicit *namespace* element shown in bold), then the DC-Terms XML syntax produced by one of the writer utilities in the Crosswalk Web Service. Note the treatment of the <subject> element, shown in bold at the bottom of Figure 4, which demonstrates that the Morfrom element with a namespace attribute is represented in the output as an element from from the DC-Simple namespace, not the DC-Terms default.

```
<record>
<header><schema name="dcterms" namespace='uri:ns:dcterms'/></header>
<field name='audience'>high school students</field>
<field name='subject' namespace='uri:ns:dc:1.1'><value>geography</value></field>
</record>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<dctermsset>
<qualifieddc xmlns dcterms="http://purl.org/dc/terms", xmlns dc=http://purl.org.dc/elements/1.1/
xsi:noNamespaceSchemaLocation="
                        http://dublincore.org/schemas/xmls/qcdc/2006/01/06/qualifieddc.xsd">
<dc:subject>geography</dc:subject>
<dcterms:audience>high school students</dcterms:audience>
</qualifieddc>
</dctermsset>
```

FIG. 4. Morfrom and native XML encodings of a DC-Terms record.

The Seel scripts in Figure 3 implement the smallest possible application profile, which contains elements from two namespaces, but this model can be extended if necessary. The DC-MARC translation at OCLC requires an additional set of elements that keep track of locally defined identifiers and other houskeeping information. To manage them, we defined elements in the *OCLC-Admin* namespace, mapped them to DC-Terms using a modified version of the spreadsheet like that shown in Figure 3, and created the corresponding Seel scripts. Now the same spreadsheet can serve as input to four translations, all easily maintained because they are automatically generated: MARC to DC-Simple, MARC to DC-Terms, MARC to OCLC-Simple, which uses the DC-Simple and OCLC-Admin namespaces; and MARC to OCLC-Terms, which uses all three namespaces.

The last operation is the override, the inverse of dumb-down. In our example, *spatial* overrides *coverage* in a DC-Terms record because it offers the chance for a more precise definition of a piece of geospatial data. The critical code is in the first map of the DCTerms2MARC script, shown on the right in Figure 3. Because <map> elements are self-contained and modular, they can carry *id* attributes that uniquely index them. Though this string can be any value, the maps in Figure 3 have straightforward names: dc-simple:1, dc-simple:2, dc-terms-1 and dc-terms-2. The map with the id value of *dc-terms:1*, which translates the *spatial* element, also has an *override* value of *dc-simple:1*, thus associating it with the map whose target is *coverage*. Now that the dominance relationship between the two maps is established, the DC-Terms map involving *spatial* is executed instead of the DC-Simple map containing *coverage* when dcterms2MARC.seel is executed.

The *dcterms:audience* element is another candidate for the override operation. When it is implemented as shown in Figure 3, it is interpreted as one of the DC-Terms elements that extends the description of DC-Simple, as we have discussed. But some members of the Dublin Core community have proposed that *dcterms:audience* should override *dc:description*. To implement this change, the metadata standards expert would need only fill in the blank box in the second row of the spreadsheet, and the *override* attribute with the appropriate value would be automatically added to the map whose id value is *dc-terms:2*, the second map in the dcTerms2MARC translation.

3. A more realistic example

Before leaving the technical discussion, we need to point out that the spreadsheet shown in Figure 3 is much too simple for realistic data. When the relationships required for managing application profiles are stripped out, the spreadsheet does little more than model a one-to-one map of source to target elements: *521 a* to *audience*, *650 a* to *subject*, and so on. More complex relationships are usually required for mapping the bibliographic metadata that pass through

OCLC's systems. For example, the mapping between source and target may be conditional on the value of data or the presence or absence of particular fields elsewhere in the record. And sometimes the data itself must be manipulated or identified with a special encoding scheme. Figure 3 shows the rest of the spreadsheet that the metadata standards expert fills out to create a production-quality MARC-to-Dublin Core translation. The greyed-out columns are the same as those in Figure 3 and indicate that the MARC tag 050 maps to *subject* regardless of whether it is interpreted as a member of the DC-Terms or DC-Simple namespace. The three columns on the right contain prompts that instruct the automated process to join the elements in MARC subfields *a* and *b* with a space and to execute this translation only if both indicators are present. The column labeled *XSI Type* specifies that the output data be interpreted as a LCC number, an encoding scheme listed in the DC-Terms namespace.

DC Simple	DC Terms	XSI Type	MARC tag	Indicators	Subfields	Special rule
Subject	Subject	dcterms:LCC	050	??	a,b	join( )

FIG. 5. An extended spreadsheet.

The Seel map created from this entry is shown in Figure 6. The additional details, shown in bold type, include the <context> element that checks for the existence of the indicators; a <value> element on the <source> that joins the subfields; and a <value> element on the <target>, whose attribute identifies the data as an LCC encoding. This example shows that even when a Seel map is expressive enough to model real-world relationships, the code remains fairly legible. The logic can be still represented transparently in a spreadsheet that is maintained by a non-programmer and automatically converted to executable code.

```
<translation>
  <header>
     <sourceschema name='marc' namespace='uri:ns:marc:21'/>
     <targetschema name='dc' namespace='uri:ns:dc:1.1'/>
  </header>
  <map id=3'>
     <source>
       <mainpath>
          <branch bid='1'><step name='050'>
            <value><join with=' ' include='a,b'></join></value>
          </branch>
       </mainpath>
       <context bid='1'>.
         <exists><path><step name="i1"/></exists>
         <exists><path><step name="i2"/></exists>
       </context>
     </source>
     <target>
       <mainpath>
          <branch bid='1'>
               <step name='subject'><value type='http://purl.org/dc/terms/LCC/'/></step>
          </branch>
       </mainpath>
     </target>
  </map>
</translation>
```

FIG. 6. The Seel map generated from the extended spreadsheet.

When this script is applied to a MARC record containing the fragment '050 ## $a PS3537.A618 $b A88 1993,' it produces the Morfrom and Dublin Core output shown in Figure 7. Note that the final outcome is a DC-Simple record, but the *xsi:type* attribute on the <subject> element properly identifies the encoding scheme of the data from the DC-Terms namespace.

```
<?xml version="1.0"?>
<record>
<header><schema name="dc" namespace="uri:ns:dc:1.1"/></header>
<field name="subject">
      <value type="http://purl.org/dc/terms/LCC">PS3537.A618 A88 1993</value>
</field>
</record>
<?xml version="1.0" encoding="UTF-8"?>
```

```
<simpledc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://dublincore.org/schemas/xmls/qdc/2006/01/06/simpledc.xsd">
<dc:subject xsi:type='http://purl.org/dc/terms/LCC'>PS3537.A618 A88 1993</dc:subject>
</simpledc>
```

FIG.7. Morfrom and native XML encodings of a DC-Simple record with a refined element.

4. Summary and future work

To summarize, we have shown how the DC-Terms application profile is invoked in a non-trivial use case: the translation of bibliographic metadata in a high-volume production environment. Users can test the results by submitting their own records to the public demo on the OCLC ResearchWorks page (OCLC, 2008). Or they can use the Dublin Core *export* functions in the OCLC Connexion® Client, as shown in Figure 8.

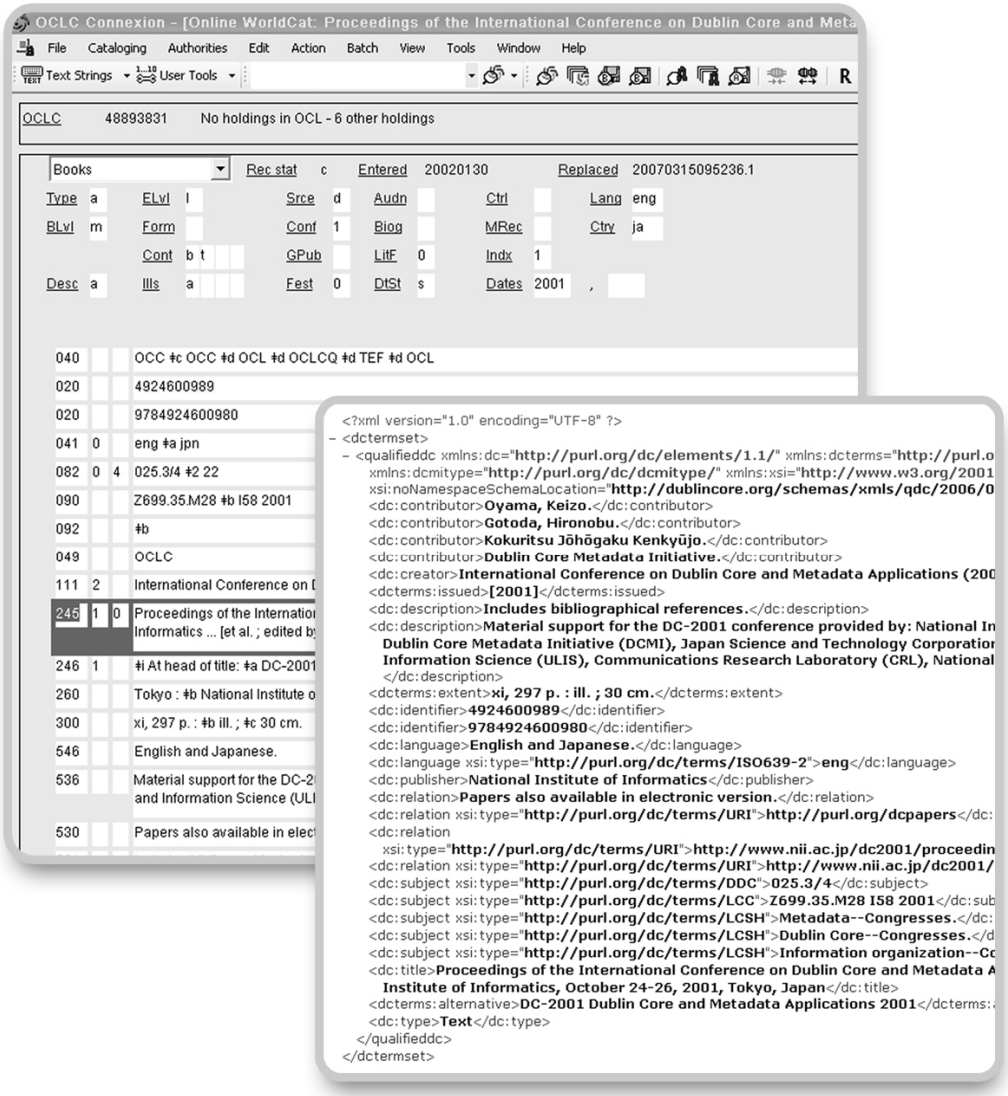


FIG. 8. A DC-Terms record crosswalked from MARC and exported to a file using OCLC Connexion Client®.

Though the result is clean, the interest is not merely theoretical because the application profile solves a significant practical problem. The complex relationships among MARC, Dublin Core, and related namespaces resolve to a mutable set of translations involving some elements that require special definitions, some that are defined in public standards, and some that are required only for local maintenance—exactly what the application profile was designed for, according to Heery and Patel (2000). With its focus on elements that can be mixed, matched, added, and redefined, the concepts that make up the application profile are a natural fit with the translation model of the Crosswalk Web Service, which attempts to achieve interoperability one element at a time by mapping those with similar meanings and manipulating their content when necessary. Before application profiles were defined and a translation model was developed that enforces transparency and reuse, the MARC-Dublin Core relationship at OCLC was managed with a large and brittle collection of pairwise translations—from DC-Terms to DC-Simple, from MARC to DC-Simple, MARC to DC-Terms, MARC to OCLC-Terms, and so on—a collection that multiplied quickly when structural variation was factored in.

Nevertheless, we consider this implementation to be the first step to a more generic solution. We eventually hope to develop a user interface that accepts input from a Web-accessible form

and produces two outputs: the executable Seel scripts, and the corresponding crosswalk formatted as a table for human consumption that is more abstract than the spreadsheet that must be maintained by metadata subject matter experts who still must be coached by our development staff. At a deeper level, we plan to exploit more of the element-oriented architecture in our translation processing, especially the URIs that are attached to each element and can carry information about the namespace it belongs to, the path to the element expressed in a formal syntax, and notes about local conditions. This is a large subject, but well worth attention because it will link our work to valuable implementations of metadata registries (Heery and Wagner, 2003) and annotation profiles (Palmér, et al., 2007). But the fact that we have achieved userful intermediate results and can envision a migration path to a more generic solution is a testament to the far-reaching consequences of Heery and Patel's original vision.

References

DCMI. (2007). *Dublin Core Collections Application Profile.* Retrieved April 10, 2008 from http://dublincore.org/groups/collections/collection-application-profile/index.shtml.

Carrier, Sarah, Jed Dube, and Jane Greenberg. (2007). The DRIADE project: Phased application profile development in support of open science. *Proceedings of the International Conference on Dublin Core & Medata Applications, 2007* (pp. 35-42).

DCMI. (2008). *DCMI metadata Terms.* Retrieved April 10, 2008, from http://dublincore.org/documents/dcmi-terms/.

GEM. (2008). *Gateway to 21st Century Skills.* Retrieved April 10, 2008, from http://www.thegateway.org/.

Gill, Tony, Anne J. Gilliland, and Mary S. Woodley. (n.d). *Introduction to metadata. Pathways to digital information.* Online Edition, Version 2.1. Retrieved June 10, 2008, from http://www.getty.edu/research/conducting_research/ standards/intrometadata/glossary.html#C.

Chan, Lois M. and Marcia Lei Zeng. (2006). Metadata interoperability and standardization - A study of methodology, Part I. *D-Lib Magazine, 12*(6). Retrieved April 10, 2008, from http://www.dlib.org/dlib/june06/chan/06chan.html.

Godby, Carol J., Devon Smith, and Eric Childress. (2008). Toward element-level interoperability in bibliographic metadata. *Code4Lib Journal, 1*(2). Retrieved April 10, 2008, from http://journal.code4lib.org/articles/54.

Greenberg, Jane, Kristina Spurgin and Abe Crystal. (2007). Functionalities for automatic-metadata generation applications: A survey of metadata experts' opinions. *International Journal of Metadata, Semantics, and Ontologies, 1*(1), 3-20.

Heery, Rachel and Manjula Patel. (2000). Application profiles: Mixing and matching metadata schemas. *Ariadne, 25*. Retrieved April 10, 2008, from http://www.ariadne.ac.uk/issue25/app-profiles/.

Heery, Rachel and Harry Wagner. (2002). A metadata registry for the Semantic Web. *D-Lib Magazine 8*(5). Retrieved June 10, 2008, from http://www.dlib.org/dlib/may02/wagner/05wagner.html.

ISO. (2008). *ISO: 2709:1996.* Retrieved April 10, 2008, from http://www.iso.org/iso/iso_catalogue/ catalogue_tc/catalogue_detail.htm?csnumber=7675

LOC (2007a). *MARC 21 specifications for record structure, character sets, and exchange media*. Retrieved April 10, 2008, from http://www.loc.gov/marc/specifications/specchartables.html.

LOC. (2007b). *MARC XML: MARC 21 Schema.* Retrieved April 10, 2008, from http://www.loc.gov/ standards/marcxml.

LOC. (2008). *MARC to Dublin Core crosswalk.* Retrieved April 10, 2008, from http://www.loc.gov/marc/marc2dc-2001.html.

OCLC. (2008). *ResearchWorks: Things to play with and think about.* Retrieved April 10, 2008, from http://www.oclc.org/research/researchworks/default.htm.

Palmér, Matthias, Fredrik Enokkson, Mikael Nilsson and Ambjörn Naeve. (2007). Annotation profiles: Configuring forms to edit RDF. *Proceedings of the International Conference on Dublin Core & Medata Applications, 2007* (pp. 10-21).

Wu, Steven, Barbara Reed and Paul Loke. (2007). SCROL application profile. *Proceedings of the International Conference on Dublin Core & Medata Applications, 2007* (pp. 22-29).

Zeng, Marcia Lei and Lois M. Chan. (2006). Metadata interoperability and standardization - A study of methodology, Part II. *D-Lib Magazine, 12*(6). Retrieved April 10, 2008, from http://www.dlib.org/dlib/june06/zeng/06zeng.html